Bi-Directional Traceability: The Hi-Five Framework Approach to Reliable Validation of Early System Designs

Martin Ouimet and Kristina Lundqvist
Embedded Systems Laboratory
Massachusetts Institute of Technology
Cambridge, MA, 02139, USA
{mouimet, kristina}@mit.edu

1. Overview

The Hi-Five framework is a holistic framework for the validation and verification of embedded real-time systems. The framework reuses the state of the art in formal verification and test case generation to provide an end-to-end solution to mitigate the typically high cost of validation and verification activities. The framework is based on a literate formal specification language, the Timed Abstract State Machine (TASM) language. The TASM language aims to capture the three key aspects of embedded real-time system behavior, namely functional behavior, timing behavior, and resource consumption. These three aspects can be captured and analyzed using the TASM language and its associated toolset [6]. Using the TASM language, the Hi-Five framework models systems at multiple levels of abstraction and provides traceability between related models. The framework provides an overarching approach to system engineering by leveraging the formal semantics of the TASM language to automate verification and test case generation.

During the early phases of system engineering, incorporating non-functional properties in system models is an approximate activity at best. For example, before an implementation exists, it is challenging to specify behavior related to time and resource consumption. Nevertheless, gaining insight into the system designs, before the system is implemented, yields considerable benefits in terms of cost and time savings. For example, evaluating design properties, such as end-to-end latency and Quality of Service can help optimize designs or select between competing designs. The TASM approach to resolving this apparent paradox is to use bi-directional traceability through levels of abstraction. During the early stages of system designs, time and resource behavior can be estimated with a high degree of uncertainty. As system models get refined all the way down to implementation, the assumptions and estimates made in higher level models become constraints on the behavior of lower level models. At each level of abstraction, analysis of timing behavior and resource consumption is performed. If undesirable behavior is found at a lower level of abstraction, the time estimates can be modified, with the changes propagating upward to higher levels of abstraction. Finally, when the system is implemented, the Hi-Five framework associates TASM models with implementation code. By performing Worst-Case Execution Time (WCET) analysis [3], exact timing metrics can be obtained and fed back through the hierarchy of models. The logical view of the approach, called bi-directional traceability, is shown in Figure 1.

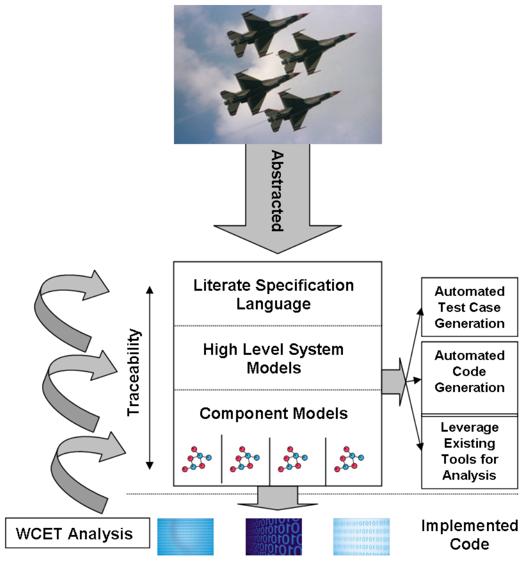
Abstracted
Literate Specification Language
High Level System Models
Component Models
Traceability
Automated Test Case Generation
Automated Code Generation
Leverage Existing Tools for Analysis
WCET Analysis
Implemented Code

Figure 1. Logical View of the TASM Approach to Traceability

2. The TASM Language

The Timed Abstract State Machine (TASM) specification language was introduced in [5], as a novel specification language for reactive real-time systems. The TASM language is based on the theory of Abstract State Machines (ASM), a method for system design that can be applied at various levels of abstraction [2]. The TASM language has formal semantics, which makes its meaning precise and enables executable specifications. The time semantics of the language revolve around the concept of durative actions. The TASM language aims to model and analyze embedded real-time systems whose aggregate behavior is defined by sets of components whose individual execution semantics can be synchronous or asynchronous. The target systems of the TASM language and the Hi-Five framework are embedded real-time controllers, such as those found in the automotive and aerospace industries. The TASM language also contains facilities for hierarchical composition, parallel composition, and synchronization channels. The TASM language uses the paradigm of durative actions to model and reason about time and resources. In TASM, execution time refers to the time that it takes to reach a certain reachable state from a start state.

3. The TASM Toolset

The TASM toolset implements the features of the TASM language through three main components - an editor, an analyzer, and a simulator. The toolset can be used during the early phases of development to understand behavior before the system is built, or it can be used throughout the development of the system to guide implementation. The type of analysis that can be performed with the toolset include verifying completeness and consistency of specifications and verifying timing properties of the specification such as the absence of deadlocks, Worst-Case Execution Time (WCET), and end-to-end latency [7]. The philosophy of the toolset is to reuse the state of the art in analytical engines to perform formal verification. The TASM toolset integrates the UPPAAL tool suite [1] to verify timing properties of TASM specifications and uses a SAT Solver to verify completeness and consistency of TASM specifications [4].

The TASM toolset includes facilities for creating and editing TASM specifications, through the TASM Editor. The editor enables the specification of functional and non-functional behavior, with standard facilities for syntax highlighting and syntax checking. Furthermore, the toolset includes facilities for executing specifications through the TASM Simulator, to visualize the dynamic behavior of the specified system. Finally, the TASM Analyzer provides analysis capabilities to verify completeness and consistency of the system, and to verify execution time of the system. The toolset is an open source project, completely written in Java; it is available, free of charge, from the TASM web site (http://esl.mit.edu/tasm).

References

[1] G. Behrmann, A. David, and K. G. Larsen. A Tutorial on UPPAAL. In *Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04)*, volume 3185 of *LNCS*. Springer-Verlag, 2004.

[2] E. Börger. The Origins and the Development of the ASM Method for High Level System Design and Analysis. *Journal of Computer Science*, 8(5), 2001.

[3] J. Engblom, A. Ermedahl, M. Nolin, J. Gustafsson, and H. Hansson. Worst-Case Execution-Time Analysis for Embedded Real-Time Systems. *International Journal on Software Tools for Technology Transfer*, 4:437–455, October 2003.

[4] M. Ouimet and K. Lundqvist. Automated Verification of Completeness and Consistency of Abstract State Machine Specifications using a SAT Solver. In *Proceedings of the 3rd International Workshop on Model-Based Testing (MBT '07), Satellite Workshop of ETAPS '07*, April 2007.

[5] M. Ouimet and K. Lundqvist. The Timed Abstract State Machine Language: An Executable Specification Language for Reactive Real-Time Systems. In *Proceedings of the 15th International Conference on Real-Time and Network Systems (RTNS '07)*, March 2007.

[6] M. Ouimet and K. Lundqvist. The Timed Abstract State Machine Toolset: Specification, Simulation, and Verification of Real-Time Systems. In *Proceedings of the 19th International Conference on Computer-Aided Verification (CAV'07)*, July 2007.

[7] M. Ouimet and K. Lundqvist. Verifying Execution Time using the TASM Toolset and UPPAAL, January 2007. Technical Report ESL-TIK-000212, Embedded Systems Laboratory, Massachusetts Institute of Technology.